НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2020р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 122 Комп’ютерні науки та інформаційні технології

на тему Система автоматизації діяльності навчально-наукових лабораторій

Виконав (-ла): студент (-ка) 4 курсу, групи ТР-62

Скирденко Тимур Юрійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник кандидат технічних наук, доцент Лабжинський В.А. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 122 Комп’ютерні науки та інформаційні технології

Спеціалізація Геометричне моделювання в інформаційних системах

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту

_____Скирденку Тимуру Юрійовичу__
(прізвище, ім’я, по батькові)

1. Тема роботи __Система автоматизації діяльності навчально-наукових лабораторій_____
_____

керівник роботи_ Лабжинський В.А., кандидат технічних наук, доцент _____
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 2020_р. № __

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи __мова програмування C#, СУБД MS Sql Server___
_____

_____

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)_____
Проаналізувати роботу навчально-наукових лабораторій, визначити архітектуру системи, розробити базу даних системи, розробити додаток____
_____
_____
_____
_____
_____
_____

5. Перелік ілюстративного матеріалу
Титульний слайд, тема та ціль роботи, задачі, які вирішуються в роботі, використані технології, Entity Framework, LINQ, архітектура системи, структура бази даних, діаграма класів рівня доступу до даних, модулі системи, результати роботи, кінцевий слайд

---

6. Дата видачі завдання ”10_”____жовтня_____2019__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	10.10.2019	
2.	Вивчення та аналіз задачі	10.10.2019-25.01.2020	
3.	Розробка архітектури та загальної структури системи	25.01.2020-28.02.2020	
4.	Розробка структур окремих підсистем	28.02.2020-06.03.2020	
5.	Програмна реалізація системи	06.03.2020-21.04.2020	
6.	Оформлення пояснювальної записки	21.04.2020-06.06.2020	
7.	Захист програмного продукту	09.06.2020	
8.	Передзахист	09.06.2020	
9.	Захист	18.06.2020	

Студент _____ _____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Записка містить 61 сторінку, 20 рисунків, 3 додатки та 10 посилань.

Мета роботи – створити систему автоматизації діяльності навчально-наукових лабораторій.

Вибір трьохрівневої архітектури додатку для роботи було зроблено, оскільки дана архітектура забезпечує, більшу масштабованість (за рахунок горизонтальної масштабованості сервера додатків і мультиплексування з'єднань) і велику конфігурованість (за рахунок ізольованості рівнів один від одного).

У результаті було розроблено систему автоматизації діяльності навчально-наукових лабораторій, яка полегшує роботу співробітників лабораторій та допомагає уникнути багатьох помилок та недоліків, які можуть виникати при роботі.

Ключові слова: трьохрівнева архітектура, система автоматизації діяльності навчально-наукових лабораторій, C#, Entity Framework, LINQ, MS Sql Server.

ABSTRACT

The note contains 61 pages, 20 pictures, 3 appendices and 10 links.

The purpose of the work is to create a system of automation of educational and scientific laboratories.

The choice of a three-tier application architecture for work was made because this architecture provides greater scalability (due to the horizontal scalability of the application server and connection multiplexing) and greater configuration (due to the isolation of levels from each other).

As a result, a system for automating the activities of educational and research laboratories has been developed, which facilitates the work of laboratory staff and helps to avoid many mistakes and shortcomings that may occur during operation.

Keywords: three-level architecture, system of automation of activity of educational and scientific laboratories, C #, Entity Framework, LINQ, MS Sql Server.

ЗМІСТ

ВСТУП

В сучасному світі перед керівниками навчально-наукових лабораторій все частіше постає питання автоматизації певних процесів. Зокрема, автоматизація ведення документації діяльності та створення менеджменту задач, які повинні бути виконані співробітниками лабораторії. Перевагами таких систем є економія часу співробітників, а також структурованість та організованість процесу виконання[1].

Навчально-наукова лабораторія є структурним підрозділом, який об'єднує фахівців з різних сфер знань. Навчально-наукова лабораторія створюється задля якісної підготовки здобувачів вищої освіти та здобувачів освітнього ступеня доктора філософії за напрямком, що відповідає профілю кафедр факультету. У своїй діяльності навчально-наукова лабораторія керується нормативно-правовими актами України в галузі освіти та науки, Статутом університету, наказами ректора, розпорядженнями проректорів за напрямками, рішеннями вчених рад університету та факультету, Науково-методичної ради університету, розпорядженнями декана факультету та протокольними рішеннями завідувача кафедри[2].

Практично будь-яка організація, яка виробляє будь-який продукт, користується послугами випробувальних лабораторій (ВЛ). У більшості випадків такі лабораторії є структурними підрозділами організацій та включаються до загальної системи управління якістю (СУЯ). Однак, поряд з вимогами до СУЯ організації, ВЛ повинна виконувати конкретні вимоги. Лабораторії беруть безпосередню участь:

- у процесі проектування та розробки нових товарів (матеріали для тестування та аналізу);

- у процесі закупівлі - контроль встановлених параметрів закупленої сировини;

- у технологічному процесі виробництва - випробування та аналіз проміжних продуктів у встановлених контрольних пунктах відповідно до технологічних регламентів, а також аналіз складу повітря в робочій зоні та навколишньому середовищі;

- у процесі моніторингу готової продукції - тестування та аналіз для

підтвердження встановлених параметрів якості продукції;

Документи, розроблені лабораторією, мають спеціальну ідентифікацію та затверджуються відповідальною лабораторією.

Розробка документів може ґрунтуватися на процесному підході, оскільки лише в цьому випадку вони будуть послідовними та продуманими, описуватимуться переходи від одного процесу до іншого, взаємодії процесів будуть правильно визначені та інформаційні потоки будуть організованим.

Процесний підхід до загального управління якістю є головним елементом нових стандартів, застосування яких передбачає встановлення чітких меж цього процесу, відповідальності, повноважень та взаємодії персоналу при виконанні певної операції. Цей принцип полягає в тому, що всі типи дій, що виконуються в організації, розглядаються як процеси, що є логічно впорядкованими послідовностями етапів різних видів роботи, які належним чином задокументовані. Застосування технологічного підходу призводить до кращого використання ресурсів, скорочення часу на виконання певної діяльності, запобігання помилок та зниження витрат на всі операції.

Дослідження проблем менеджменту роботи ТОВ “ТС Експертиза” сприяло розробці програмного продукту, який вирішує проблеми автоматизації роботи навчально-наукової лабораторії.

Звіт з переддипломної практики містить 5 розділів.

У першому розділі описується постановка задачі системи автоматизації роботи навчально-наукових лабораторій.

У другому розділі описуються підходи до вирішення проблеми автоматизації діяльності навчально-наукових лабораторій.

У третьому розділі вказуються основні засоби розробки даної системи.

У четвертому розділі дано опис реалізованого програмного продукту та його архітектури.

У п’ятому розділі описано роботу користувача з програмною системою.

1 ПОСТАНОВКА ЗАДАЧІ АВТОМАТИЗАЦІЇ ДІЯЛЬНОСТІ НАВЧАЛЬНО-НАУКОВИХ ЛАБОРАТОРІЙ

Основним «продуктом» лабораторії є інформація. Вона подається у формі даних та документів. Документи та дані - це елементи системи якості, за допомогою яких здійснюється комунікація як в лабораторії, так і за її межами.

Згідно відомостей та теоретичного матеріалу, який був наданий керівником практики та керівником дипломної роботи, розробити інформаційну систему автоматизації діяльності навчально-наукової лабораторії. Система має складатися з двох модулів:

— модуль зберігання та швидкого пошуку документації лабораторії;

— модуль моніторингу виконання задач.

В модулі зберігання та швидкого пошуку документації необхідно було врахувати можливість додавати до бази нові документи, типи документів, співробітників, обладнання лабораторії та навчальні кафедри. При додаванні нових даних повинні проводитися певні перевірки на їх коректність та правильність. Після завантаження всіх даних користувач повинен мати змогу групувати, сортувати документи за їх типом, датою створення та модифікації, автором документу. Усі документи користувач може переглядати, редагувати та зберігати за допомогою програмного засобу Microsoft Word 2016.

В модулі моніторингу виконання задач необхідно надати користувачу можливість заносити основний план робіт до бази та записувати до Word-файлу додаткові плани. Створення системи документації складається з документально підтверджених процедур для забезпечення якості діяльності лабораторії. Спеціальний документ повинен встановлювати порядок розробки, затвердження, реєстрації, публікації, впровадження, перевірки, обліку, зміни, перегляду, розширення та скасування документів, на підставі яких проводяться випробування. Для отримання точної картини та подальшого аналізу всю документацію лабораторії

треба розділити на дві частини: внутрішні та зовнішні документи.

Робоча система для лабораторії є важливим інструментом підтримки та підвищення конкурентоспроможності, постійного вдосконалення лабораторії та джерела економічної вигоди. Особливе місце в системі якості лабораторії посідає управління записом даних, що, з одного боку, дає можливість для постійного вдосконалення своєї діяльності, а з іншого - підвищує продуктивність.

Лабораторія повинна мати наступну документацію:

— положення про лабораторію, затверджене керівником установи;
— паспорт лабораторії, затверджений керівником установи;
— свідоцтво(а) про атестацію / акредитацію;
— ліцензія на відповідний вид діяльності, видана відповідно до чинного законодавства (для лабораторій недержавної власності);
— організаційно-розпорядча документація - накази, інструкції та інші документи, що регламентують діяльність лабораторії;
— нормативна документація, яка регламентує вимоги до об'єктів дослідження та методів дослідження;
— документація для системи забезпечення якості досліджень;
— інструкція з якості;
— інструкції з внутрішнього та зовнішнього контролю якості досліджень;
— інструкції з протиепідемічного режиму, охорони праці та безпеки праці;
— документи на обладнання та засоби вимірювальної техніки:
— реєстраційні документи на обладнання (журнал, картки тощо);
— паспорт на кожну одиницю обладнання та вимірювальної техніки;
— документація щодо працівників лабораторії;
— посадові інструкції;
— документи з питань підвищення кваліфікації та сертифікації персоналу (сертифікати тощо);

Документована інформація може бути представлена на різних носіях інформації та в різних формах: паперові документи, комп'ютерні файли, програми, графіка, діаграми, відеозаписи тощо. У лабораторній практиці термін «лабораторна

документація» частіше застосовується стосовно цих елементів.

Для забезпечення доступності, надійності, своєчасності та достовірності результатів робіт лабораторну документацію слід суворо контролювати. Цього можна досягти, лише створивши ефективну систему управління документами. Система може бути реалізована різними способами: електронною системою управління документами, системою «ручного управління» або їх комбінацією. Все залежить від фінансових можливостей лабораторії та рівня компетентності персоналу. З будь-яким варіантом реалізації існує низка питань, які необхідно опрацювати перед впровадженням системи. До таких питань належать: структура лабораторної документації; види документації; діяльність з управління документами.

Для роботи з даними було створено базу даних за допомогою системи управління реляційними базами даних Microsoft SQL Server 2017. Безпосередньо інформаційна система написана мовою програмування C#. Графічний інтерфейс розроблено з використанням інтерфейсу програмування додатків Windows Forms. Метою розробки інформаційної системи автоматизації діяльності навчально-наукових лабораторій є програмний продукт, який вирішить проблему ведення документації, мінімізує використання нестійких до зараження комп'ютерними вірусами флеш-накопичувачів інформації, а також збереже час співробітників лабораторії, завдяки автоматизації цих задач.

2 АНАЛІЗ МЕТОДІВ ВИРІШЕННЯ ПРОБЛЕМИ АВТОМАТИЗАЦІЇ ДІЯЛЬНОСТІ НАВЧАЛЬНО-НАУКОВИХ ЛАБОРАТОРІЙ

На сьогоднішній день проблема автоматизації діяльності навчально-наукових лабораторій залишається актуальною. Вирішенням цієї задачі займається багато математиків та програмістів, проте досі не було розроблено додатку, який був би актуальним для більшості лабораторій. Існує багато математичних постановок задачі, проте усі вони різняться вхідними параметрами, а отже не є універсальними. Існуючі додатки представлені лише на комерційних ринках. Проте через різні умови праці, обмеження та вхідні дані для кожної лабораторії багатьом з них немає сенсу купувати такі додатки. Ці програмні засоби не вирішують конкретні проблеми роботи певної навчально-наукової лабораторії.

Для вирішення проблеми автоматизації діяльності навчально-наукових лабораторій пропонують використовувати додатки, створені на основі багаторівневої, клієнт-серверної, мікросерверної, мікроядерної архітектури.

2.1. Розподілені системи

Розподілена система - це група комп'ютерів, яку користувачі бачать як єдину, єдину систему.

Існує 2 аспекти розподіленої системи:

— всі комп'ютери автономні;

— користувачам така система здається уніфікованою.

Розподілені програми в Інтернеті, як правило, базуються на моделі типу клієнт-сервер - у цій моделі програми структуруються шляхом їх поділу на серверні процеси, що надають спеціалізовані послуги для клієнтських процесів, тоді як серверні процеси можуть працювати з одним або кількома клієнти:

Хоча і клієнт, і сервер теоретично можуть проживати на одному комп'ютері, в основному системи цієї архітектури запускають клієнтські процеси на одному комп’ютері, а серверні процеси на іншому та використовують мережеві з'єднання для обміну інформацією. Така модель дозволяє одному процесу працювати незалежно від інших, виконувати певні завдання та розподіляти обчислювальне навантаження.

Зазвичай клієнт - це настільний ПК, на якому працює програмне забезпечення для кінцевих користувачів - це будь-які прикладні програми, які надсилають запит по мережі на сервер і обробляють інформацію, отриману від нього. Наприклад, ПК, який працює під управлінням Windows та запускає клієнтську програму, може надіслати запит на сервер, який працює на іншій системі або на сервері баз даних. Сервер, отримавши запит, виконує певні дії від імені клієнта.

Взаємодія між клієнтом та сервером в Інтернеті здійснюється за допомогою запитів, які клієнт надсилає на сервер, та відповідей, які сервер надсилає на запит клієнта.

Дослідження різних інтернет-додатків показують, що в 70 відсотках випадків для виконання певних дій користувачеві не потрібно звертатися до сервера - всі ці дії можуть бути реалізовані на стороні клієнта, якщо їх виконання може бути запрограмовано на ньому.

2.2. Клієнт-серверна архітектура

На сьогоднішній день найбільш популярним серед розробників вважається клієнт-серверна архітектура додатків. Клієнт-серверна архітектура набула своєї популярності завдяки динамічному розвитку мережі Інтернет та зосередження значної частини інформації в базах даних на серверах. Сервер у цьому випадку вважається абстрактною машиною в мережі, здатною приймати HTTP-запит, обробляти його та повертати відповідь. У контексті цієї роботи його фізична природа та внутрішня архітектура взагалі не важливі, чи це студентський ноутбук

чи величезний кластер промислових серверів, розкиданих по всьому світу. Головне, щоб сервер відповідав головному правилу - почути, зрозуміти і надати відповідь.

Клієнтом також може бути все, що може генерувати та надсилати HTTP-запит. Клієнтом може бути сценарій JavaScript, який працює в браузері, мобільний додаток, що працює на сервері.

Клієнт-серверну архітектуру можна означити, як концепцію інформаційної мережі в якій основна частина її ресурсів зосереджена в серверах, обслуговуючих своїх клієнтів. Така архітектура визначає такі типи компонентів:

1. Набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
2. Набір клієнтів, які використовують сервіси, що надаються серверами;
3. Мережа, яка забезпечує взаємодію між клієнтами та серверами[3].

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов’язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

— рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;
— прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;
— рівень управління даними, який забезпечує зберігання даних та доступ до них.

Під клієнт-серверним додатком ми розуміємо інформаційну систему, засновану на використанні серверів баз даних. Загальний вигляд інформаційної системи в архітектурі клієнт-сервер показаний на малюнку.

На стороні клієнта виконується код програми, який обов'язково включає компоненти, які підтримують інтерфейс кінцевого користувача, виробляють звіти та виконують інші функції, що стосуються додатків.

Клієнтська частина додатка взаємодіє з клієнтською частиною програмного

забезпечення для управління базами даних, яка, власне, є окремим представником СУБД для програми.

Знову ж таки, проявляються недоліки в термінології. Зазвичай, коли компанія оголошує про випуск наступного сервера баз даних, явно розуміється, що в цьому продукті є і клієнтська складова. Поєднання клієнтська частина сервера баз даних здається дещо дивним, але нам доведеться використовувати цей термін.

Інтерфейс між клієнтською частиною програми та клієнтською частиною сервера баз даних, як правило, заснований на використанні мови SQL. Тому такі функції, як, наприклад, попередня обробка форм, призначених для запитів у базу даних, або генерування отриманих звітів, виконуються в коді програми.

Нарешті, клієнтська частина сервера баз даних, використовуючи доступ до мережі, отримує доступ до сервера баз даних, передаючи їй текст оператора мови SQL.

Як правило, компанії, що виробляють розширені сервери баз даних, прагнуть до того, щоб їх продукцію можна було використовувати не тільки в сьогоднішніх стандартних TCP/IP-орієнтованих мережах, але і в мережах на основі інших протоколів (наприклад, SNA або IPX/SPX). Тому при організації мережевих взаємодій між клієнтською та серверною частинами СУБД часто використовуються не стандартні інструменти високого рівня (наприклад, механізми програмної розетки або виклики віддалених процедур), а власні функціонально схожі інструменти, менш залежні від особливості протоколів мережевого транспорту.

Коли ми говоримо про інтерфейс, заснований на мові SQL, ми повинні усвідомлювати, що незважаючи на титанічні зусилля щодо стандартизації цієї мови, не існує такої реалізації, в якій стандартні засоби мови не розширюватимуться. Необережне використання розширень мови робить програму повністю залежною від конкретного виробника сервера баз даних.

У продуктах майже всіх компаній сервер отримує від клієнта текст оператора в SQL. Сервер компілює отриманий оператор. Головне, що в будь-якому випадку, виходячи з інформації, що міститься в таблицях каталогу баз даних, непроцедурне представлення оператора перетворюється на певну процедуру його виконання. Далі

(якщо компіляція пройшла успішно), оператор виконується.

Оператор може належати до класу операторів для визначення (або створення) об'єктів бази даних (було б точніше і правильніше говорити про елементи схеми бази даних або про об'єкти метабаз даних). Зокрема, можна визначити домени, таблиці, обмеження цілісності, тригери, привілеї користувача, збережені процедури. У будь-якому випадку, коли виконується оператор створення елемента схеми бази даних, відповідна інформація розміщується в таблицях каталогу баз даних (у таблицях метабаз даних). Обмеження цілісності зазвичай зберігаються в метабазі безпосередньо в текстовому вікні. Для дій, визначених у тригерах та збережених процедурах, процедурний виконуваний код формується та зберігається у таблицях каталогу. Обмеження цілісності, тригери та збережені процедури є певним чином представниками програм у базі даних, що підтримується сервером; вони складають основу серверного боку програми.

При виконанні операторів вибірки даних отриманий набір даних формується на основі вмісту таблиць, на які впливає запит і, можливо, з використанням індексів, що підтримуються в базі даних (ми навмисно не використовуємо тут термін результуюча таблиця, тому що залежно від конкретного типу оператора, результат може бути впорядкований, а таблиці, тобто відносини невпорядковані за визначенням). Серверна частина СУБД відправляє результат клієнтської частини, а остаточна обробка вже виконується в клієнтській частині програми.

При виконанні операторів зміни вмісту бази даних (INSERT, UPDATE, DELETE) перевіряється, що обмеження цілісності, визначені в цьому пункті (ті, що належать до класу негайно перевірених), не будуть порушені, після чого виконується відповідна дія (супроводжується шляхом зміни всіх відповідних індексів та змін журналу). Далі сервер перевіряє, чи впливає ця зміна на стан тригера будь-якого тригера, і якщо такий тригер виявлений, виконує процедуру своєї дії. Ця процедура може включати додаткові заяви про зміну бази даних, які можуть викликати інші тригери тощо. Ми можемо припустити, що ті дії, які виконуються на сервері баз даних при перевірці задоволення обмежень цілісності та коли спрацьовують тригери, є діями серверної частини програми .

При виконанні операторів для зміни схеми бази даних (додавання або видалення стовпців існуючих таблиць, зміна типу даних існуючого стовпця існуючої таблиці тощо) також можуть спрацювати тригери, тобто, іншими словами, серверна частина програми може бути виконана.

Так само тригери можуть спрацьовувати при знищенні об'єктів схеми бази даних (доменів, таблиць, обмежень цілісності тощо). Спеціальний клас операторів SQL складається з операторів викликів попередньо визначених та збережених процедур у базі даних. Якщо збережена процедура визначена з використанням достатньо розробленої мови, що включає як непроцедурні оператори SQL, так і суто процедурні конструкції (наприклад, Oracle PL/SQL), то серйозна частина програми може бути розміщена в такій процедурі, яка буде виконується на стороні сервера, а не на стороні клієнта. Коли оператор завершує транзакцію, сервер повинен перевірити відповідність усім так званим відкладеним обмеженням цілісності (такі обмеження включають обмеження, накладені на вміст всієї таблиці бази даних або на декілька таблиць одночасно). Знову ж таки, перевірка відкладених обмежень цілісності може трактуватися як виконання зворотного кінця програми. Як бачите, в організації клієнт-сервер клієнти можуть бути досить тонкими, а сервер повинен бути досить товстим, щоб можна було задовольнити потреби всіх клієнтів.

З іншого боку, розробники та користувачі інформаційних систем, що базуються на архітектурі клієнт-сервер, часто незадоволені постійно існуючими накладними мережевими накладними витратами, які виникають у зв'язку з необхідністю зв'язуватися з клієнтом із сервером при кожному наступному запиті. На практиці поширена ситуація, коли для ефективної роботи окремого клієнтського компонента інформаційної системи насправді потрібна лише незначна частина загальної бази даних. Це призводить до ідеї підтримки локального кешу спільної бази даних на стороні клієнта.

Насправді, концепція кешування локальних баз даних є окремим випадком концепції реплікаційних (або, як їх іноді називають, тиражованих) баз даних. Як і в загальному випадку, для підтримки локального кешу бази даних програмне забезпечення робочої станції повинно містити компонент управління базою даних -

спрощену версію сервера баз даних, яка, наприклад, може не забезпечувати багатокористувацький режим доступу. Окремою проблемою є забезпечення узгодженості кешів та загальної бази даних. Тут можливі різні рішення - від автоматичної підтримки узгодженості за рахунок базового програмного забезпечення для управління базами даних до повної передачі цього завдання на прикладний рівень. У будь-якому випадку, клієнти стають товщими, незважаючи на те, що сервер не тонший.

Сформулюємо деякі попередні висновки. Архітектура клієнт-сервер на перший погляд здається набагато дорожчою, ніж архітектура файлового сервера. Потрібні більш потужне обладнання (принаймні для сервера) та значно більш досконалі засоби управління базами даних. Однак це лише частково правда. Величезною перевагою архітектури клієнт-сервер є її масштабованість і здатність розвиватися в цілому. При розробці інформаційної системи на основі цієї архітектури слід приділяти більше уваги грамотності загальних рішень. Технічні засоби пілотної версії можуть бути мінімальними (наприклад, одна з робочих станцій може використовуватися як апаратна основа сервера баз даних).

Загалом існує багато різних типів архітектур, які успішно застосовуються. Однією з найбільш використовуваних є класична трирівнева система, яка передбачає поділ програми на три рівні.

Тут треба сказати відразу, що багаторівнева архітектура часто позначає два не зовсім пов'язані між собою поняття: n-layer та n-tier. I layer, і tier зазвичай позначаються словом "рівень", іноді слово “layer” також використовується стосовно "шару". Однак в обох випадках рівні будуть різного порядку.

Tier представляє фізичний шар. Тобто, якщо ми говоримо про трирівневу архітектуру, то n-ярусну програму можна було б розділити на такі рівні: сервер бази даних, веб-додаток на веб-сервері та браузер користувача. Тобто кожен рівень представляв би спеціальний окремий фізичний процес, навіть якщо і сервер бази даних, і веб-сервер, і браузер користувача були б на одному комп’ютері. Якщо мобільний додаток альтернативно використовувався як клієнт, то це був би ще один фізичний рівень.

Layer представляє логічний рівень. Тобто у нас може бути рівень доступу до даних, рівень бізнес-логіки, рівень презентації, рівень обслуговування тощо. У цьому випадку логічні рівні не збігаються з фізичними. Так, звичайно рівень презентації в додатку ASP.NET містить як контролери, які обробляють введення, так і представлення, що відображаються у веб-браузері, тобто поділяється на два фізичні рівні.

У цьому випадку ми поговоримо конкретно про логічні рівні, тобто про архітектуру n-шарів.

Класична трирівнева система складається з таких рівнів (рисунок 2.1):

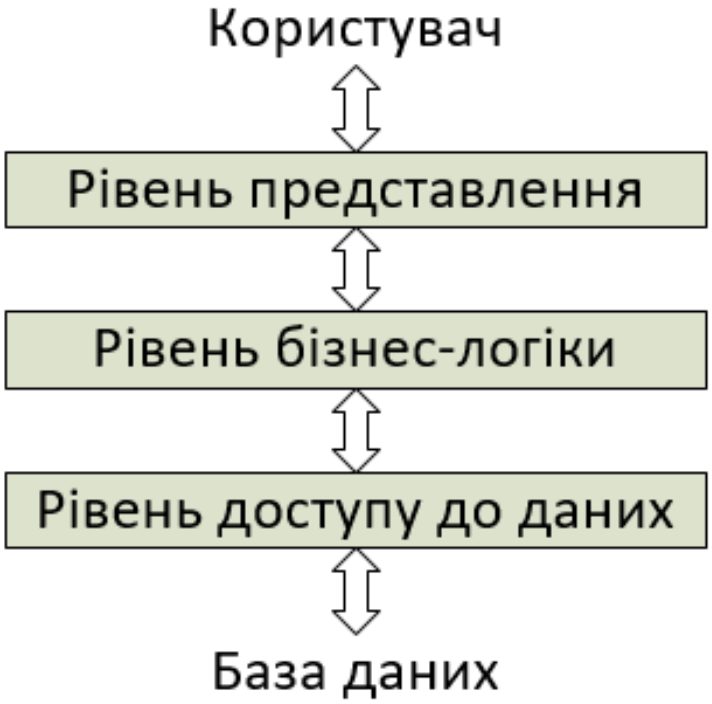

Рисунок 2.1 — Рівні трьохрівневої системи

Рівень представлення: це рівень, з яким користувач безпосередньо взаємодіє. Цей рівень включає компоненти інтерфейсу користувача, механізм отримання вводу від користувача. Що стосується asp.net mvc, на цьому рівні є перегляди та всі ті компоненти, які складають користувальницький інтерфейс (стилі, статичний html, сторінки JavaScript), а також переглядають моделі, контролери, запитують контекстні об’єкти.

Бізнес-рівень (рівень бізнес-логіки): містить набір компонентів, які відповідають за обробку даних, отриманих від рівня представлень, реалізує всю необхідну логіку програми, всі обчислення, взаємодіє з базою даних та передає результат обробки на рівень презентації.

Рівень доступу до даних: зберігає моделі, що описують використовувані об'єкти, також існують конкретні класи для роботи з різними технологіями доступу до даних, наприклад, контекстний клас даних Entity Framework. Тут також зберігаються сховища, через які рівень логіки бізнесу взаємодіє з базою даних.

Слід зазначити, що крайні рівні не можуть взаємодіяти один з одним, тобто рівень презентації (стосовно ASP.NET MVC, контролерів) не може безпосередньо отримати доступ до бази даних і навіть рівня доступу до даних, але лише через рівень бізнес-логіки .

Рівень доступу до даних не залежить від інших рівнів, рівень ділової логіки залежить від рівня доступу до даних, а рівень подання - на рівні бізнес-логіки.

Компоненти, як правило, повинні бути вільно зв'язані (нещільне з'єднання), тому введення залежностей є невід'ємною частиною багаторівневих додатків.

Що означає ASP.NET MVC, насамперед, стосовно рівня презентації, тоді як решта рівнів можуть бути реалізовані незалежно та можуть використовуватися в додатках, що використовують інші технології, такі як Windows Forms, WPF тощо. І, як правило, вся програма в цілому представлятиме рішення у Visual Studio, а окремі рівні представлятимуть проекти. У той же час, неправильно вважати, що якщо рівень обов'язково повинен відповідати окремому проекту. При необхідності ми можемо розділити один рівень на кілька проектів, головне, щоб його функціональність представляла єдину логічну ланку.

2.3. Особливості системи автоматизації роботи ТОВ “ТС Експертиза”

В процесі дослідження роботи ТОВ “ТС Експертиза” виявилося, що весь робочий процес фіксується в єдиній внутрішній CRM системі Creatio.

CRM-система Creatio включає комплекс готових онлайн-інструментів і процесів для різних галузей і моделей бізнесу. Можливості системи включають: управління бізнес-процесами — їх проектування, автоматизацію, аналітику; управління клієнтською базою; планування та управління продажами; управління маркетинговими кампаніями; автоматизацію діловодства та документообігу; управління робочим часом; контроль виконання доручень; відстеження результатів роботи та аналітику. Робота з даними Creatio CRM в режимі off-line забезпечується наявністю розширення Bpm'online Outlook Connector.

Платформа Creatio розроблена з використанням трирівневої архітектури на базі .NET Framework 4. Для створення користувацьких інтерфейсів використовуються технології HTML, AJAX та Microsoft Silverlight. У Creatio застосовується сервіс-орієнтована архітектура (Service-oriented architecture, SOA), що базується на службах.

Створення додатків відбувається не за допомогою написання нового програмного коду, а за допомогою зв'язування сервісів. В основі платформи Creatio лежать бізнес-процеси. Для реалізації нової функціональності або зміни користувальницьких інтерфейсів потрібно вибрати стандартні елементи і створити в редакторі бізнес-процес, який реалізує необхідну бізнес-логіку. Створені процеси побудовані за нотації BPMN 2.0 (Business Process Modeling Notation). Для забезпечення безпеки і розмежування прав доступу в Creatio забезпечується можливість налаштування прав доступу по ролях, підтримка криптографічного протоколу SSL, обмеження доступу до об'єктів, полів об'єктів, записів[4].

3 ЗАСОБИ РОЗРОБКИ

Вибір мови програмування та технологій, які застосовуються при розробці програмного продукту, напряму залежить від задачі, яку необхідно вирішити[5].

Середовищем розробки програмного продукту було обрано Microsoft Visual Studio 2019.

Графічний інтерфейс розроблявся за допомогою інтерфейса програмування додатків Windows Forms.

Код написано на об’єктно-орієнтовній мові програмування C#.

Для формування задач і їх збереження у Word-файл була використана бібліотека Microsoft.Office.Interop.Word.

Для створення бази даних було використано Microsoft SQL Server 2017.

Для зв’язку з базою даних було використано бібліотеки System.Data.Entity.

3.1 Середовище розробки Visual Studio 2019

Microsoft Visual Studio - це лінійка продуктів Microsoft, що включає інтегроване середовище розробки програмного забезпечення та ряд інших інструментів. Ці продукти дозволяють розробляти як консольні програми, так і програми GUI, включаючи ті, що підтримують технологію Windows Forms, а також веб-сайти, веб-додатки та веб-сервіси як у власних, так і керованих кодах для всіх платформ, що підтримуються Windows, Windows CE, .NET Framework, Windows Mobile, Xbox, Windows Phone .NET Compact Framework та Silverlight.

Visual Studio включає редактор вихідного коду, який підтримує технологію IntelliSense та можливість легко перетворювати код. Вбудований налагоджувач може працювати як відладчик рівня джерела або як відладчик рівня машини. Інші вбудовані інструменти включають редактор форм для спрощення створення графічного інтерфейсу програми, веб-редактора, дизайнера класів та дизайнера схем бази даних. Visual Studio дозволяє створювати та підключати сторонні додатки

(плагіни) для розширення функціональності майже на кожному рівні, включаючи додавання підтримки для систем управління версіями вихідного коду (таких як Subversion і Visual SourceSafe), додаючи нові набори інструментів (наприклад, для редагування та коду візуального дизайну на доменних мовах програмування) або інструментів для інших аспектів процесу розробки програмного забезпечення (наприклад, клієнт Team Explorer для роботи з Team Foundation Server).

Visual Studio - це інструментальне середовище розробки, яке включає інтегроване середовище розробки, редактор вихідного коду та інтегрований налагоджувач. Багато інших інструментів можна отримати, підключивши плагіни - сторонні розширення. Його вибрали через доступність інтеграції з Unity 3d, підтримку мови програмування C #, наявність можливості писати спеціальні тести Unit. Це також платформа, найбільш вивчена розробником, яка дозволяє швидко розробляти додатки, не витрачаючи часу на вивчення нового матеріалу.

Visual Studio — це повністю інтегроване середовище розробки. Воно спроектоване таким чином, щоб робити процес написання коду, його налагодження та компіляції в збірку для поставки кінцевим споживачам якомога простішим[6]. Основні можливості Visual Studio 2019:

1. Текстовий редактор, в якому можна писати програми на різних мовах програмування, зокрема C#. Редактор тексту автоматично проставляє табуляцію та потрібні відступи, що полегшує читання коду, закриває дужки, формує блоки коду, виділяє ключові слова різним кольором.
2. Візуальний редактор форм, який автоматично додає до основного коду програми код елементів управління.
3. Інтегрований відладник, який дозволяє ставити точки початку та кінця відладки не виходячи із середовища розробки.
4. Довідкова система MSDN, до якої можна звертатися із середовища розробки. Досить виділити незрозуміле слово або помилку в коді, натиснути F1 і система покаже потрібні розділи довідки.
5. Доступ до інших програм, зокрема SQL Server, який допомагає переглядати дані в таблицях та налагоджувати зв’язок з базами даних.

3.2 Windows Forms

Для створення графічних інтерфейсів за допомогою платформи .NET застосовуються різні технології – Windows Forms, WPF, додатки для магазину Windows Store. Проте найбільш простою та зручною платформою досі залишається Windows Forms або форми[7].

Windows Forms використовуються, зокрема для розробки десктопних додатків. Їх перевагою є те, що вони стабільно працюють без доступу в Інтернет та доступаються до ресурсів на комп'ютері більш безпечним шляхом.

Поява додатку до нас переважно через форми. Форми - основні складові. Вони забезпечують контейнер для різних елементів управління. А механізм подій дозволяє елементам форми реагувати на введення користувача та таким чином взаємодіяти з користувачем.

Коли ми відкриваємо проект у Visual Studio у графічному редакторі, ми можемо побачити візуальну частину форми - ту частину, яку ми бачимо після запуску програми та куди ми передаємо елементи з панелі управління. Але насправді форма ховає потужний функціонал, що складається з методів, властивостей, подій тощо

Форма — це візуальна поверхня, на якій відображається інформація для користувача. Коли користувач намагається зробити щось з елементом форми, додаток реагує на цю дію та генерує подію, яка обробляється за допомогою коду, який прописаний для кожного елемента.

Для роботи з базами даних найбільш популярним є елемент управління DataGridView. Він відображає табличні дані у класичному форматі в вигляді колонок та рядків, де кожна комірка містить дані певного формату. Завдяки DataGridView можна налаштувати відображення даних в окремих комірках.

За допомогою сервісів Windows Forms можна дуже легко налаштувати зв'язок з базами даних та працювати з таблицями.

Windows Forms використовує механізм подій для взаємодії з користувачем. Події в Windows Forms являють собою стандартні події C #, які застосовуються

лише до візуальних компонентів і відповідають тим же правилам, що і події в C #. Але створення обробників подій у Windows Forms все ще має деякі особливості.

Перш за все, WinForms має деякий стандартний набір подій, який здебільшого доступний для всіх візуальних компонентів. Окремі елементи додають власні події, але принципи роботи з ними будуть подібними.

Керування - це візуальні класи, які отримують введення користувача та можуть викликати різні події. Усі елементи управління успадковуються від класу Control і тому мають ряд загальних властивостей:

— **Anchor**: визначає, як елемент буде розтягуватися;

— **BackColor**: визначає колір тла елемента;

— **BackgroundImage**: визначає фонове зображення елемента;

— **ContextMenu**: контекстне меню, яке відкриється при натисканні правою кнопкою миші на елемент. Визначено за допомогою елемента ContextMenu;

— **Cursor**: Представляє, як з'являється курсор миші, коли ви наводите курсор на елемент;

— **Dock**: Вказує розташування елемента на формі;

— **Enabled**: визначає, чи буде елемент доступний для використання. Якщо ця властивість хибна, елемент блокується;

— **Font**: Встановлює шрифт тексту для елемента;

— **ForeColor**: Визначає колір шрифту;

— **Location**: отримує верхній лівий кут елемента керування;

— **Name**: Назва елемента;

— **Size**: Вказує розмір елемента;

— **Width**: ширина елемента;

— **Height**: висота елемента;

— **TabIndex**: визначає порядок переходу елемента за допомогою натискання клавіші Tab;

— **Tag**: Дозволяє зберегти значення, пов'язані з цим елементом управління.

3.3 Бібліотека Microsoft.Office.Interop.Word

Бібліотека Microsoft.Office.Interop.Word у середовищі Visual Studio полегшує роботу с Word-файлами.

Головним є інтерфейс _Application. Він містить налаштування та параметри, наприклад багато опцій у вікні «Параметри», а також властивості, які повертають об’єкти вищих рівнів, наприклад ActivCell, ActiveSheet та інші. Це основний інтерфейс в COM-класі, який потрібен керованому коду для взаємодії з відповідним COM-об'єктом. Використовується цей первинний інтерфейс, тільки якщо метод, який використовується, має те ж ім'я, що й подію COM-об'єкта; в цьому випадку приведення до цього інтерфейсу для виклику методу і приведення до інтерфейсу останніх подій для підключення до події. В іншому випадку використовується інтерфейс .NET, похідний від COM-класу COM, для доступу до методів, властивостей і подій COM-об'єкта.

3.4 Бібліотека System.Linq

LINQ (Language-Integrated Query) надає просту та зручну мову для запиту до джерела даних. Об'єктом, який реалізує інтерфейс IEnumerable (наприклад, стандартні колекції, масиви), DataSet, XML-документ, може виступати джерелом даних. Але незалежно від типу джерела, LINQ дозволяє застосовувати однаковий підхід до всіх вибірок даних. Існує кілька різновидів LINQ:

— **LINQ to Objects**: використовується для роботи з масивами та колекціями;

— **LINQ to Entities**: використовується під час доступу до баз даних за допомогою технології Entity Framework;

— **LINQ to Sql**: технологія доступу до даних у MS SQL Server;

— **LINQ to XML**: використовується під час роботи з XML-файлами;

— **LINQ для DataSet**: використовується під час роботи з DataSet;

— **Parallel LINQ (PLINQ)**: використовується для виконання паралельних запитів.

На додаток до стандартного синтаксису from .. in .. select, ми можемо використовувати спеціальні методи розширення, визначені для інтерфейсу IEnumerable для створення запиту LINQ. Зазвичай ці методи реалізують таку ж функціональність, як і оператори LINQ, такі як where або orderby.

Список використаних методів розширення LINQ:

— **Select**: визначає проекцію вибраних значень;
— **Where**: визначає фільтр вибору;
— **OrderBy**: сортування елементів у порядку зростання;
— **OrderByDescending**: сортування елементів у порядку зменшення;
— **ThenBy**: Вказує додаткові критерії впорядкування елементів у порядку зростання;
— **ThenByDescending**: встановлює додаткові критерії сортування елементів у порядку зменшення;
— **Join**: з'єднує дві колекції за певним атрибутом;
— **GroupBy**: групує елементи за ключем;
— **ToLookup**: групує елементи за ключем, всі елементи додаються до словника;
— **GroupJoin**: одночасно з'єднує колекції та групи елементів за клавішею;
— **Reverse**: упорядковує елементи в зворотному порядку;
— **All**: визначає, чи всі предмети колекції відповідають певній умові;
— **Any**: визначає, задовольняє хоча б одному елементу колекції до певної умови;
— **Contains**: визначає, чи містить колекція конкретний предмет;
— **Distinct**: видаляє дублікати елементів із колекції;
— **Except**: повертає різницю двох колекцій, тобто тих елементів, які створені лише в одній колекції;
— **Union**: поєднує дві однорідні колекції;
— **Intersect**: повертає перетин двох колекцій, тобто тих елементів, які зустрічаються в обох колекціях;
— **Count**: підраховує кількість предметів колекції, які відповідають певній

умові;

— **Sum**: підраховує суму числових значень у колекції;

— **Average**: обчислює середнє число числових значень у колекції;

— **Min**: знаходить мінімальне значення;

— **Max**: знаходить максимальне значення;

— **Take**: вибирає певну кількість елементів;

— **Skip**: пропускає певну кількість елементів;

— **TakeWhile**: повертає ланцюжок елементів послідовності, доки умова справжня;

— **SkipWhile**: пропускає елементи в послідовності, поки вони не задовольняють заданій умові, а потім повертає решту елементів;

— **Concat**: поєднує дві колекції;

— **Zip**: поєднує дві колекції відповідно до конкретної умови;

— **First**: вибирає перший елемент колекції;

— **FirstOrDefault**: вибирає перший елемент колекції або повертає значення за замовчуванням;

— **Single**: вибирає один елемент колекції, якщо колекція містить більше або менше одного елемента, викидається виняток;

— **SingleOrDefault**: вибирає перший елемент колекції або повертає значення за замовчуванням;

— **ElementAt**: вибирає елемент послідовності в певному індексі;

— **ElementAtOrDefault**: вибирає елемент колекції за певним індексом або повертає значення за замовчуванням, якщо індекс знаходиться поза діапазоном;

— **Last**: вибирає останній елемент колекції;

— **LastOrDefault**: вибирає останній елемент у колекції або повертає значення за замовчуванням;

3.5 Бібліотека System.Data.Entity

Entity Framework являє спеціальну об'єктно-орієнтовану технологію на базі фреймворка .NET для роботи з даними.

Центральної концепцією Entity Framework є поняття сутності або entity. Сутність представляє набір даних, асоційованих з певним об'єктом. Тому дана технологія передбачає роботу не з таблицями, а з об'єктами і їх наборами.

Entity Framework передбачає три можливі способи взаємодії з базою даних:

— Database first: Entity Framework створює набір класів, які відображають модель конкретної бази даних;

— Code first: розробник створює клас моделі даних, які будуть зберігатися в бд, а потім Entity Framework за цією моделлю генерує базу даних і її таблиці.

До .NET розробники часто використовували для написання коду ADO.NET або блоку доступу до корпоративних даних для збереження або отримання даних програми з базової бази даних. Використовували для того, щоб відкрити з'єднання з базою даних, створити набір даних для отримання або подання даних до бази даних, перетворення даних з DataSet в .NET-об'єкти або навпаки, щоб застосувати бізнес-правила. Це був громіздкий і схильний до помилок процес. Корпорація Майкрософт створила структуру під назвою "Entity Framework" для автоматизації всіх цих заходів, пов’язаних із базою даних для вашої програми.

Entity Framework - це рамка ORM з відкритим кодом для програм .NET, що підтримуються Microsoft. Це дозволяє розробникам працювати з даними, використовуючи об’єкти класів, що належать до домену, не орієнтуючись на основні таблиці та колонки бази даних, де зберігаються ці дані. За допомогою Entity Framework розробники можуть працювати на більш високому рівні абстракції, коли вони обробляють дані, і можуть створювати та підтримувати орієнтовані на дані програми з меншим кодом порівняно з традиційними програмами.

Офіційне визначення: Entity Framework - це об'єктно-реляційний маппер (O/RM), який дозволяє розробникам .NET працювати з базою даних за допомогою

об'єктів .NET. Це виключає необхідність більшості коду доступу до даних, який розробникам зазвичай потрібно писати.

На рисунку 3.1 показано, де Entity Framework вписується у вашу програму.

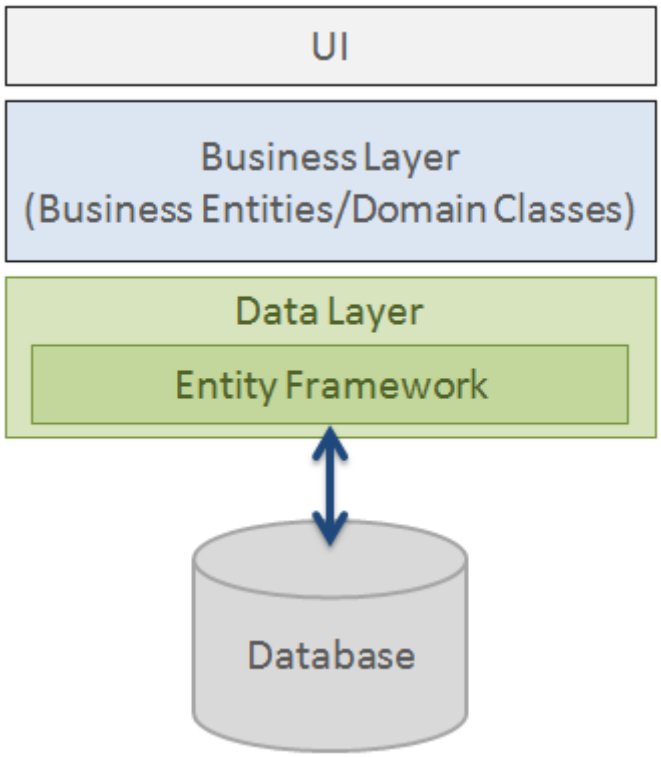

Рисунок 3.1 — Entity Framework в структурі додатку

Згідно з наведеним вище малюнком, Entity Framework знаходиться між доменними класами та базою даних. Він зберігає дані, що зберігаються у властивостях об’єктів, а також витягує дані з бази даних та автоматично перетворює їх на об'єкти.

Особливості Entity Framework:

— **Cross-platform**: EF Core - це крос-платформа, яка може працювати в Windows, Linux та Mac;

— **Modelling**: EF (Entity Framework) створює EDM (модель даних особи) на основі об'єктів POCO (Plain Old CLR Object) з властивостями get/set різних типів даних. Вона використовує цю модель під час запиту або збереження даних сутності до базової бази даних;

— **Querying**: EF дозволяє нам використовувати запити LINQ (C#/VB.NET) для отримання даних з бази даних. Постачальник баз даних переведе цей

LINQ-запит на специфічну мову запитів (наприклад, SQL для реляційної бази даних). EF також дозволяє нам виконувати необроблені SQL запити безпосередньо в базі даних;

— **Change Tracking**: EF відслідковує зміни, що відбулися у випадках, коли ваші сутності (Значення властивостей) потрібно подати до бази даних.
— **Saving**: EF виконує в базі даних команди INSERT, UPDATE та DELETE на основі змін, які відбулися у ваших об'єктах під час виклику методу SaveChanges(). EF також забезпечує асинхронний метод SaveChangesAsync();
— **Concurrency**: EF використовує оптимістичну паралельність за замовчуванням, щоб захистити зміни, внесені іншим користувачем з моменту отримання даних із бази даних;
— **Transactions**: EF виконує автоматичне управління транзакціями під час запитів або збереження даних. Він також пропонує варіанти налаштування управління транзакціями;
— **Caching**: EF включає перший рівень кешування з коробки. Отже, повторний запит поверне дані з кешу, а не звернення до бази даних.
— **Built-in Conventions**: EF дотримується конвенцій щодо схеми програмування конфігурації та включає набір правил за замовчуванням, які автоматично налаштовують модель EF;
— **Configurations**: EF дозволяє нам налаштувати модель EF за допомогою атрибутів анотації даних або Fluent API для зміни умовних умов.
— **Migrations**: EF надає набір команд міграції, які можна виконати на консолі диспетчера пакетів NuGet або інтерфейсі командного рядка для створення або управління базовою схемою бази даних;

3.6 Microsoft SQL Server

SQL Server — центральна частина платформи обробки даних Microsoft. Він є однією з найбільш популярних систем управління базами даних (СУБД) в світі.

Дана СУБД вирішує питання управління даними для проектів різних масштабів: від невеликих додатків до високонавантажених систем[9].

Центральним аспектом в MS SQL Server, як і в будь-який СУБД, є база даних. База даних являє сховище даних, організованих певним способом. Нерідко фізично база даних представляє файл на жорсткому диску, хоча таке відповідність необов'язково. Для зберігання і адміністрування баз даних застосовуються системи управління базами даних (database management system) або СУБД (DBMS). І якраз MS SQL Server є однією з такою СУБД.

Для організації баз даних MS SQL Server використовує реляційну модель. Ця модель баз даних була розроблена ще в 1970 році Едгаром Коддом. А на сьогоднішній день вона фактично є стандартом для організації баз даних.

Реляційна модель передбачає зберігання даних у вигляді таблиць, кожна з яких складається з рядків і стовпців. Кожен рядок зберігає окремий об'єкт, а в стовпчиках розміщуються атрибути цього об'єкта.

Для ідентифікації кожного рядка в рамках таблиці застосовується первинний ключ (primary key). В якості первинного ключа може виступати один або декілька стовпців. Використовуючи первинний ключ, ми можемо посилатися на певну рядок в таблиці. Відповідно два рядки не можуть мати один і той же первинний ключ.

Через ключі одна таблиця може бути пов'язана з іншого, тобто між двома таблицями можуть бути організовані зв'язку. А сама таблиця може бути представлена у вигляді відносини (“relation”).

Для взаємодії з базою даних застосовується мова SQL (Structured Query Language). Клієнт (наприклад, зовнішня програма) відправляє запит на мові SQL за допомогою спеціального API. СУБД належним чином інтерпретує і виконує запит, а потім посилає клієнту результат виконання.

Склад даних - це база даних, що представляє собою сукупність таблиць із набраних стовпців. SQL Server підтримує різні типи даних, включаючи основні, такі як Integer, Float, Decimal, Char, Varchar, Binary, Text та інші.

Статистика серверів доступна у вигляді віртуальних таблиць і представлень. Крім таблиць, база даних може містити й інші об'єкти, включаючи представлення

даних, процедури, індекси та обмеження, а також журнал транзакцій. База даних SQL Server може містити максимум 231 об'єкт і може охоплювати кілька файлів на рівні операційної системи з максимальним розміром файлу 260 байт. Дані в базі даних зберігаються в первинних файлах даних з розширенням .mdf. Вторинні файли даних, які були ідентифіковані з розширенням .ndf, використовуються таким чином, що дані однієї бази даних можуть бути розповсюджені в більш ніж один файл і, можливо, в більш ніж одну файлову систему. Файли журналу ідентифікуються з розширенням .ldf.

Дисковий простір бази даних поділяється на послідовно пронумеровані сторінки, кожна по 8 Кб. Сторінка - основний блок вводу/виводу для операцій на SQL Server. Сторінка позначена 96-байтовим заголовком, який зберігає метадані про сторінку, включаючи номер сторінки, тип сторінки, вільний простір на сторінці та ідентифікатор об'єкта, якому вони належать. Тип сторінки визначає дані, що містяться на сторінці: дані, що зберігаються в базі даних, індекс, карта розповсюдження, карта змін, яка містить інформацію про зміни, внесені до інших сторінок з моменту останнього резервного копіювання або реєстрації, або містять великі типи даних, наприклад зображення або текст.

Для фізичного зберігання таблиці його ряди діляться на ряд розділів (пронумеровані від 1 до N). Розмір розділу визначається користувачем; за замовчуванням всі рядки знаходяться в одному розділі. Таблиця розділена на кілька розділів для розподілу бази даних на кластери. Рядки у кожному розділі зберігаються у вигляді дерева B або купи.

Основний спосіб отримання даних із бази даних SQL Server - це запит. Запит виражається за допомогою варіанту SQL, який називається T-SQL. У запиті декларативно вказується, що слід отримати. Він обробляється процесором запитів, який виявляє послідовність кроків, необхідних для отримання необхідних даних. Послідовність дій, необхідних для завершення запиту, називається планом запиту. Існує кілька способів обробляти один і той же запит. Наприклад, для запиту, що містить оператора вибору та оператора об'єднання, спочатку виконується об'єднання обох таблиць, а потім вибір або навпаки. У цьому випадку SQL Server вибирає план,

який, як очікується, буде швидшим. Оптимізація запитів виконується безпосередньо в процесорі запитів.

Native Client SQL Server - одна з технологій доступу до даних у базі даних SQL Server. Вирішуючи, чи використовувати власного клієнта SQL Server як технологію доступу до даних, слід враховувати декілька факторів. Якщо ви використовуєте мову програмування з керованим кодом, наприклад Microsoft Visual C# або Visual Basic, і вам потрібно отримати доступ до нових функцій SQL Server, то для нових програм потрібно використовувати постачальника даних .NET Framework для SQL Server, яка є частиною .NET Framework. Якщо ви розробляєте додаток на основі COM і вам потрібен доступ до нових функцій SQL Server, вам слід використовувати власний клієнт SQL Server. Якщо доступ до нових функцій SQL Server не потрібен, ви можете продовжувати використовувати компоненти WDAC. Для існуючих програм OLE DB та ODBC найважливішим питанням є необхідність доступу до нових функцій SQL Server. Якщо у вас є налагоджена програма, яка не потребує нових функцій SQL Server, ви можете продовжувати використовувати компоненти WDAC. Але якщо вам потрібен доступ до нових функцій, таких як новий тип даних XML, вам потрібно використовувати Native Client SQL Server. Компоненти Native Client SQL Server та MDAC підтримують рівень ІЗ ЗАПРОШЕНОГО ОБРАЗУВАННЯ транзакцій при використанні версійних версій, але виділення транзакцій знімків підтримується лише нативним клієнтом SQL Server. З точки зору програмування рівень ізоляції транзакції READ COMMITTED з версією версії такий же, як транзакції READ COMMITTE

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

В даному розділі йдеться про процес програмування додатку. Описуються різні етапи створення програмного продукту: від проектування бази даних в Microsoft SQL Server до інтерфейсу.

4.1 Створення бази даних в Microsoft SQL Server

Всього у Microsoft SQL Server було створено 10 таблиць:

— Employee;

— Department;

— Position;

— User;

— Document;

— DocumentType;

— Equipment;

— EquipmentType;

— UsrTask;

— UsrTaskType.

В усіх таблицях наявне поле id, якому надано властивості первинного ключа. Первиний ключ — це один з прикладів унікального індексу і використовується для унікальної ідентифікації записів таблиці. У даному випадку первинні ключі надані перш за все для ідентифікації записів та полегшення подальшої роботи з ними, зокрема для видалення помилкових або непотрібних записів.

Сутність видалення — у визначенні рядків, які мають бути видалені. Дія DELETE завжди визначається її предикатом WHERE. Якщо забрати WHERE, то видалиться уся таблиця. Задаючи предикат WHERE, ми можемо зменшити область дії DELETE до певної групи рядків або до одного рядка. При видаленні одного рядка він зазвичай ідентифікується за допомогою первинного ключа[10].

Наявність первинного ключа також є важливою, тому що він використовується для зв’язку таблиць між собою. Сама структура бази даних виглядає наступним чином(рисунок 4.1):

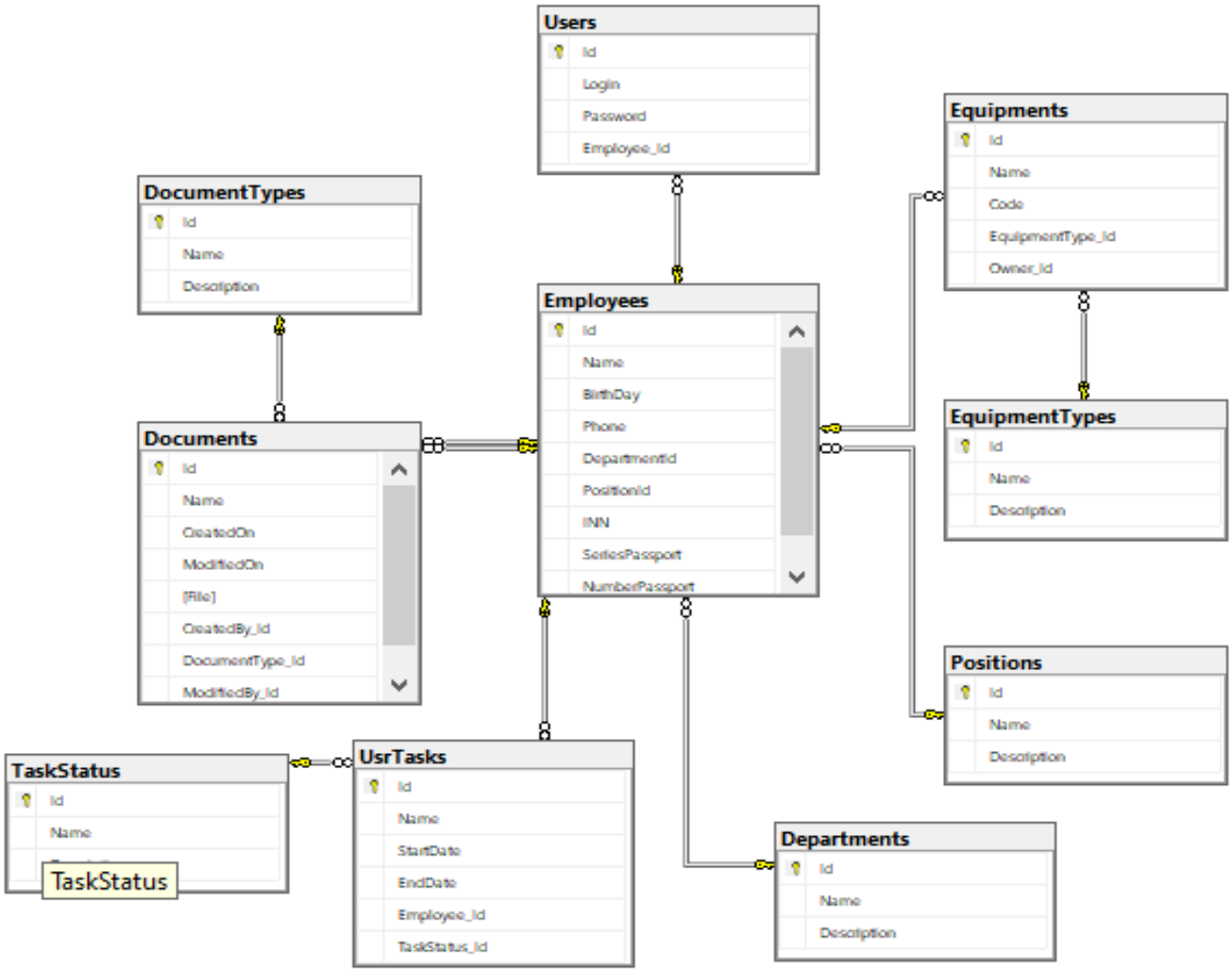

Рисунок 4.1 — Структура бази даних

В таблиці Documents 8 полів:

— Id;

— Назва;

— Дата створення;

— Дата модифікації;

— Файл документу;

— Створив;

— Змінив;

— Тип документу.

Поле Дата створення та Дата модифікації призначене для збереження дати створення та модифікації файлу відповідно та для забезпечення можливості пошуку та сортування файлів.

Поле Файл документу — поле двійкового типу, в якому і зберігається сам файл документу.

Поле Створив та Змінив зберігає в собі Id користувача системи, тобто користувача, яким була здійснена авторизація в системі і котрий створив чи редагував запис з файлом чи сам файл. Наразі усі документи зберігаються в одній таблиці, але для кожного документа додано поле, в якому вказано, до типу він відноситься. Структуру таблиці Subject наведено нижче (рисунок 4.2).

Имя столбца	Тип данных	Разрешить значения NULL
Id	int	☐
Name	nvarchar(MAX)	☑
CreatedOn	datetime	☐
ModifiedOn	datetime	☐
[File]	varbinary(MAX)	☑
CreatedBy_Id	int	☑
DocumentType_Id	int	☑
ModifiedBy_Id	int	☑

Рисунок 4.2 — Структура таблиці Documents

В таблицы Employees відображаються всі співробітники лабораторії. Вона містить наступні поля:

— id:

— Ім'я;

— Дата народження;

— Телефон;

— Навчальна кафедра;

— Посада;

— ІНФО;

— Серія паспорта;

— Номер паспорта;

Структуру таблиці Employees наведено нижче (рисунок 4.3).

Имя столбца	Тип данных	Разрешить значения NULL
Id	int	☐
Name	nvarchar(MAX)	☑
BirthDay	datetime	☐
Phone	nvarchar(MAX)	☑
DepartmentId	int	☑
PositionId	int	☑
INN	nvarchar(MAX)	☑
SeriesPassport	nvarchar(MAX)	☑
NumberPassport	nvarchar(MAX)	☑

Рисунок 4.3 — Структура таблиці Employees

Структуру таблиць Equipments, UsrTask, наведено на рисунку 4.4.

Equipments

Имя столбца	Тип данных
Id	int
Name	nvarchar(MAX)
Code	nvarchar(MAX)
EquipmentType_Id	int
Owner_Id	int

UsrTasks

Имя столбца	Тип данных
Id	int
Name	nvarchar(MAX)
StartDate	datetime
EndDate	datetime
Employee_Id	int
TaskStatus_Id	int

Рисунок 4.4 — Структура таблиць

4.2 Створення проекту Visual Studio

Проект містять об'єкти, необхідні для створення програми у Visual Studio, такі як файли вихідного коду, растрові карти, піктограми та посилання на компоненти та служби. При створенні проекту, Visual Studio створює рішення, яке буде містити проекти. Після цього при необхідності можна додати інші нові або існуючі проекти.

Рішення можуть також містити файли, не пов'язані з конкретним проектом. На рисунку 4.5 зображено створене рішення.

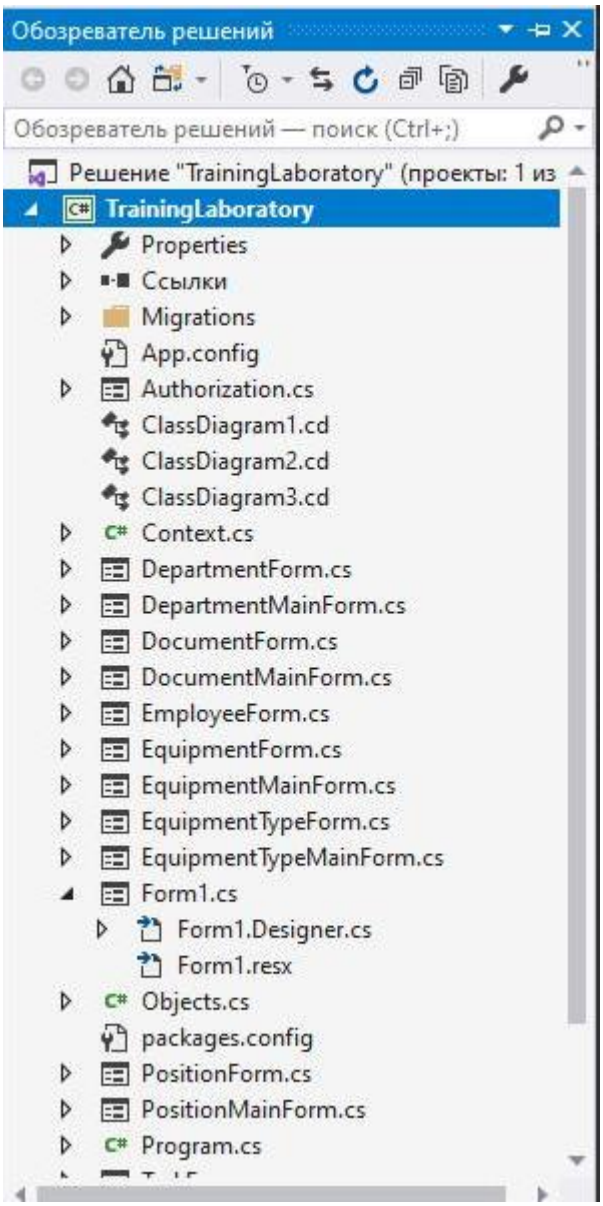

Рисунок 4.5 — Рішення TrainingLaboratory

4.3 Підключення бази даних до проекту Visual Studio та робота з таблицями

Функціональність Entity Framework Core для роботи з MS SQL Server заснована на класах, які розташовані в просторі імен Microsoft.EntityFrameworkCore. Серед усього набору класів цього простору імен слід виділити наступне:

— DbContext: визначає контекст даних, який використовується для взаємодії з базою даних;

— DbSet/DbSet <TEntity>: представляє набір об'єктів, які зберігаються в базі даних;

— DbContextOptionsBuilder: встановлює параметри з'єднання;

У додатку, що працює з базою даних через Entity Framework, потрібен контекст (клас, похідний від DbContext).

У контекстному класі даних колекція об'єктів представляє клас DbSet <T>.

Крім того, щоб налаштувати з'єднання, треба перекрити метод OnConfiguring. Параметр класу DbContextOptionsBuilder, переданий йому за допомогою методу UseSqlServer, дозволяє налаштувати рядок з'єднання для підключення до MS SQL Server. У цьому випадку в якості сервера буде використовуватися двигун localdb (Server = (localdb)\\mssqllocaldb), а файл бази даних буде називатися systemDb (Database = systemDb).

І також варто зазначити, що за замовчуванням немає бази даних. Тому в конструкторі контекстного класу визначається виклик методу Database.EnsureCreate (), який при створенні контексту автоматично перевіряє наявність бази даних і, якщо вона відсутня, створює її.

Завдяки засобам роботи Visual Studio 2019 з базами даних, створених за допомогою SQL Server базу даних було додано до джерела даних проекту.

На різних формах проекту “Редагування співробітників”, “Редагування документів”, “Редагування посад” та “Редагування задач” було створено відповідні DataGridView, які відображають зміст таблиць на форму та дозволяють додавати нові записи та редагувати та видаляти наявні.

На формі “Документи” було створено DataGridView, який відображає дані, які вже збережені до бази даних, але не дозволяє додавати нові записи. Додавати записи до DataGrivView можна за допомогою елементів управління button, при цьому відкривається нова форма, де в елементах textbox та comboBox вводяться необхідні дані. Дані, які відображаються в comboBox витягуються з відповідних таблиць бази даних. В коді елементів управління використовуються Linq-запити.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

В цьому розділі приведені системні вимоги для роботи з додатком та сценарії роботи користувача з ним.

5.1 Системні вимоги

Яку б програму ви не запускали, знати її системні вимоги дуже важливо з наступних причин.

Основна з них - вимоги до операційної системи. Багато чудових додатків минулого більше не можна запускати в сучасних операційних системах, таких як Windows 10. Вони просто не підтримуються. Вам пощастило, якщо у додатку є оновлений перевипуск, але якщо його там немає, вам доведеться встановити іншу ОС, особливо для вашої програми.

Друга причина - вимоги до відеокарти та процесора. Їх потужність щороку зростає, але новітні програми все ж більш вимогливі, ніж комп'ютери користувачів. Це особливо важливо, якщо ви використовуєте старі компоненти, і ви довго не оновлювались.

Третє - це кількість оперативної пам’яті та постійної пам’яті. Важливий фактор, тому що якщо у вас недостатньо оперативної пам’яті, то додаток працюватиме з перервами, тому вам доведеться придбати ще 16-32 гігабайти. І якщо на жорсткому диску немає місця, програма навіть не почне встановлювати, поки ви не звільнить місце. Ну або не купуйте жорсткий диск 4-8 терабайт. Деякі програми не працюють краще на класичних жорстких дисках, але на покращених твердотілих SSD - швидкості завантаження карти збільшуються і навіть FPS збільшується.

Четверте - вимога швидкості Інтернету, що вкрай важливо для мережевого додатку. Зазвичай вимоги до ширини каналу пишуть самі виробники - це може бути 512/1024Кб/с.

Але якщо сервери додатків розташовані далеко від вашого місця проживання, то краще мати оптичний канал зв'язку, який дозволяє зменшити ping до 30-50 мс.

Для певного програмного забезпечення розрізняють мінімальні та рекомендовані системні вимоги.

Мінімальні системні вимоги - це сукупність умов, необхідних для запуску та експлуатації програмного продукту. Однак наявність мінімальних системних вимог не виключає можливості роботи програмного забезпечення на комп'ютерах, які слабкіші за характеристиками, ніж мінімальні.

Рекомендовані системні вимоги - сукупність характеристик, що передбачають оптимальну роботу більшості функцій виробу. Однак, навіть якщо комп'ютер відповідає рекомендованим системним вимогам, це не означає високої продуктивності програмного забезпечення, наприклад, у деяких іграх неможливо грати при максимальних налаштуваннях графіки.

Для забезпечення коректної та безвідмовної роботи інформаційної системи автоматизації діяльності навчально-наукових лабораторій персональний комп’ютер повинен мати процесор не гірше, ніж Intel ® Pentium ® / Celeron ® / Xeon™ або з тактовою частотою не менше 1,8 GHz або AMD 6 / Turion ™ / Athlon ™ / Duron ™ / Sempron ™ для користувачів процесорів від фірми AMD.

Комп’ютеру користувача повинно бути доступно не менше 2 Gb оперативної пам’яті та графічне ядро не гріше, ніж Intel ® HD Graphics 2000, що еквівалентно графічним картам з об’ємом пам’яті не менше, ніж 128 Mb.

На комп’ютері повинна бути встановлена система Windows 10, Windows 7, Windows XP Service Pack 2 або Windows Vista Service Pack 2.

На жорсткому диску повинно бути не менше, ніж 30 Mb вільного місця для завантаження програмного продукту, а також збереження документів Word із задачами.

Користувачу також знадобиться програма із сімейства Microsoft Office, а саме – Microsoft Office Word. Вона потрібна для перегляду документів.

5.2 Робота з програмним продуктом

Початкове вікно Windows Forms для роботи з додатком “Система автоматизації діяльності навчально-наукових лабораторій” являє собою вікно авторизації і має наступний вигляд (рисунок 5.1). Авторизація дозволяє ідентифікувати відвідувача додатку та одночасно обмежувати права доступу до певних його ресурсів та можливостей стороннім користувачам.

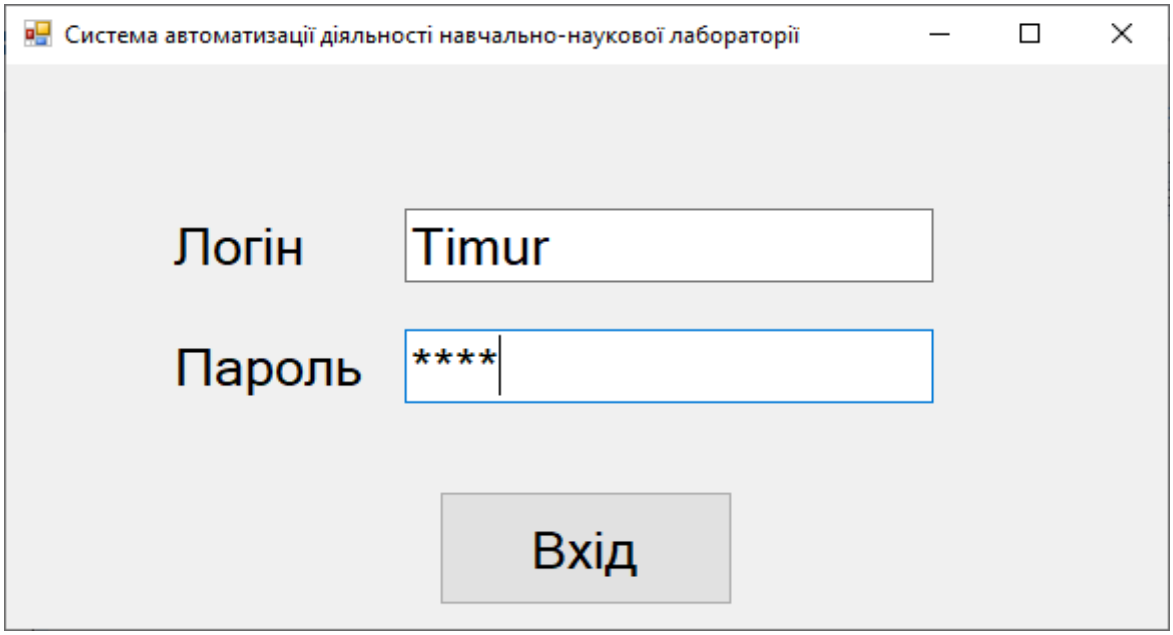

Рисунок 5.1 — Вікно авторизації Windows Forms

Користувач повинен ввести Логін та Пароль, які він отримає від адміністратора системи. Помилка авторизації - неправильний логін, пароль та інші дані. Вводячи кодові слова, користувач повинен звернути увагу на правильний порядок символів, регістр, встановлену розкладку клавіатури. Якщо авторизація не вдається, система блокує відвідувача доступу до системи і може виконувати такі дії:

— фіксація факту несанкціонованого доступу;
— звуковий або світловий сигнал, повідомлення на екрані;
— обмежений доступ на певний час;
— пропонувати код повторного набору;
— відновлення паролю.

Після успішної авторизації користувач перейде до головного вікна системи, де він може вже переглянути список всіх співробітників лабораторії, а також перейти до вікон: навчальні кафедри, посади, обладнання, документи (рисунок 5.2).

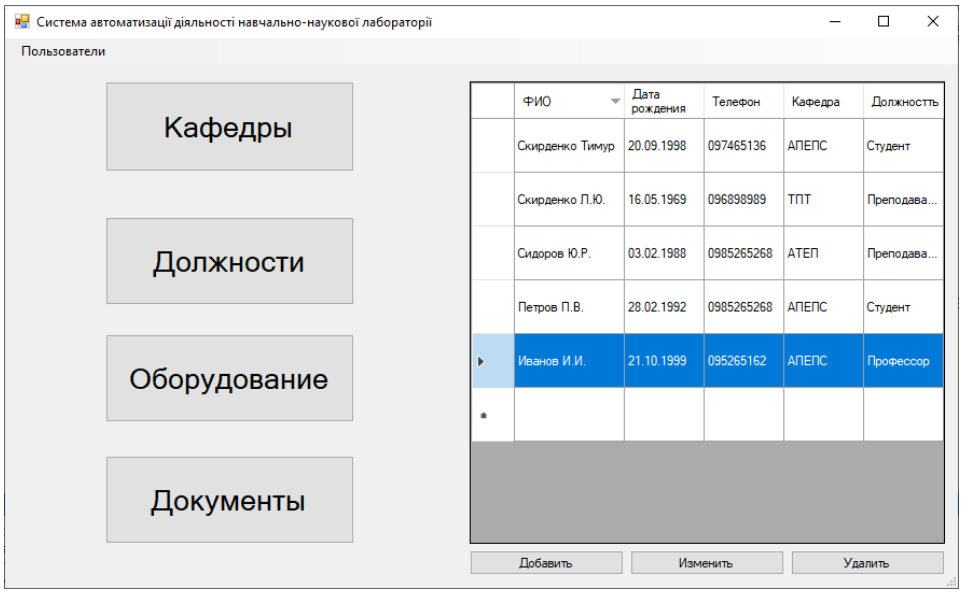

Рисунок 5.2 — Головне вікно системи

В цьому вікні користувач може додати нового співробітника, видали наявного, продивлятися рядки з записами та зберегти зміни за допомогою елементів управління. При натисканні на кнопку “Додати” відкривається додаткове вікно (рисунок 5.3), де треба вести всю інформацію про співробітника.

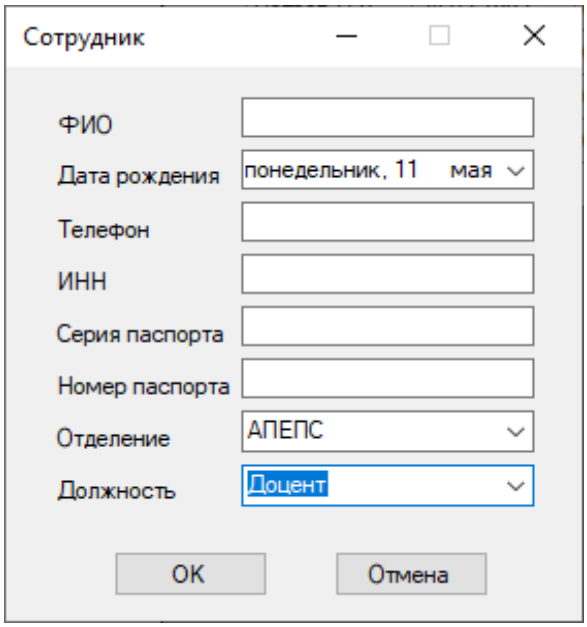

Рисунок 5.3 — Вікно додавання нового співробітника

Для редагування співробітника треба виділити потрібний запис і натиснути “Змінити”. В результаті відкриється вікно (рисунок 5.4).

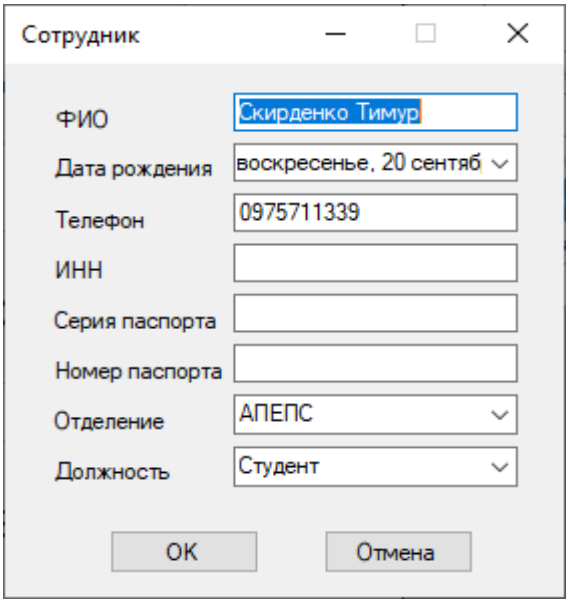

Рисунок 5.4 — Вікно редагування співробітника

Для видалення запису достатньо її виділити і натиснути “Видалити”.

В верхній частині вікна є кнопка Користувачі, доступ до якої має лише користувач з логіном Адміністратор.

У системі є два типи користувачів: звичайний та адміністратор. Різновидом прав адміністратора є спеціальний захист від небажаних змін налаштувань, які може здійснити недосвідчений користувач. Ці зміни можуть призвести до збоїв у системі або нестабільності системи.

Адміністратор може переглядати, додавати, редагувати та видаляти інших користувачів системи. Якщо ваш обліковий запис не має прав адміністратора, ви все одно можете нормально працювати з системою. Однак це не дасть вам повного доступу до системи, а лише дозволить виконати обмежену кількість дій.

Натиснувши на кнопку “Документи” користувач відкриває вікно додатку, де може переглядати, додавати, редагувати і видаляти документи з системи, це вікно представлено на рисунку 5.5.

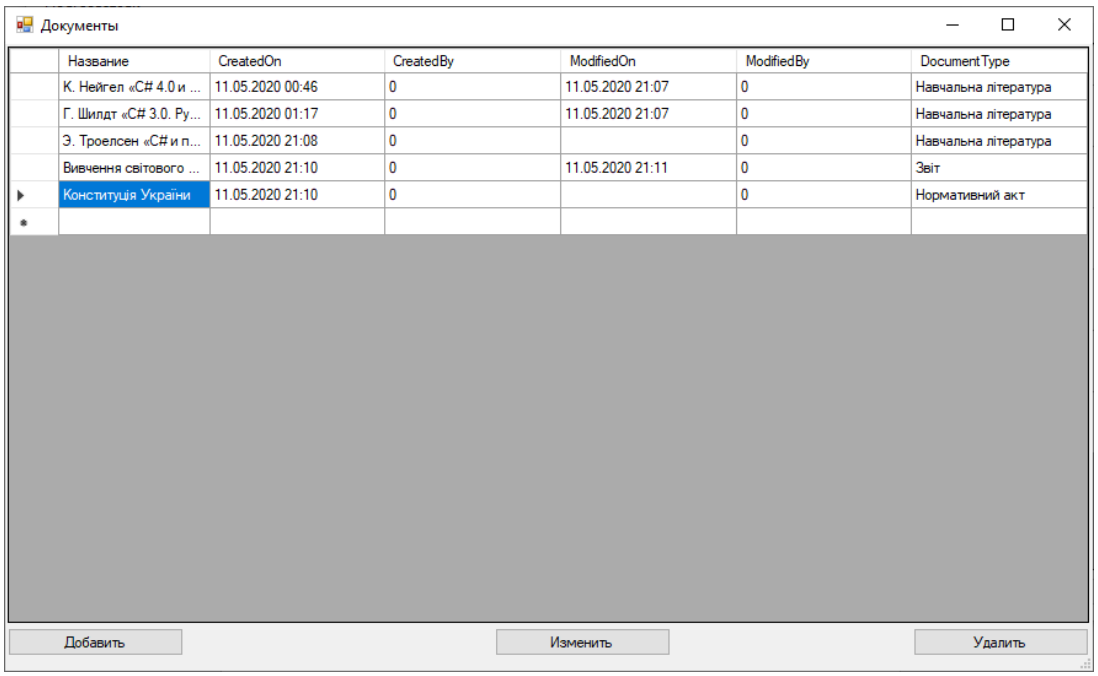

Рисунок 5.5 — Вікно “Документи”

Натиснувши кнопку Додати користувач потрапляє до вікна, зображеного на рисунку 5.6.

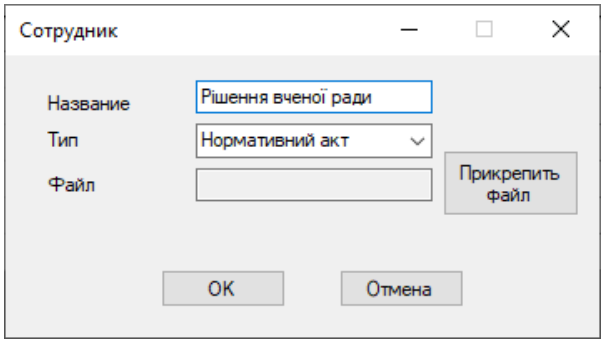

Рисунок 5.6 — Вікно редагування документів

Після натискання кнопки ОК користувачу відображається повідомлення, що документ створено (рисунок 5.7). Також користувач має змогу сортувати документи за назвою, типом, датою створення чи редагування, а також пошук за автором документу.

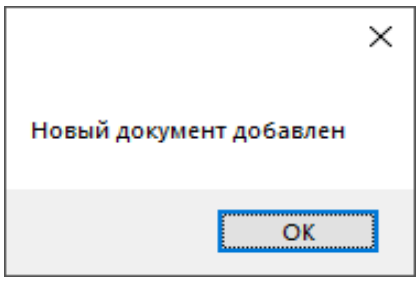

Рисунок 5.7 — Повідомлення про успішне створення документу в системі

Створена система передбачає зберігання даних про наявне в лабораторії обладнання. Це є одним з важливих аспектів систем даного типу, оскільки облік обладнання завжди присутній в навчально-наукових лабораторіях.
Для роботи з даними про обладнання треба перейти до вікна Обладнання (рисунок 5.8).

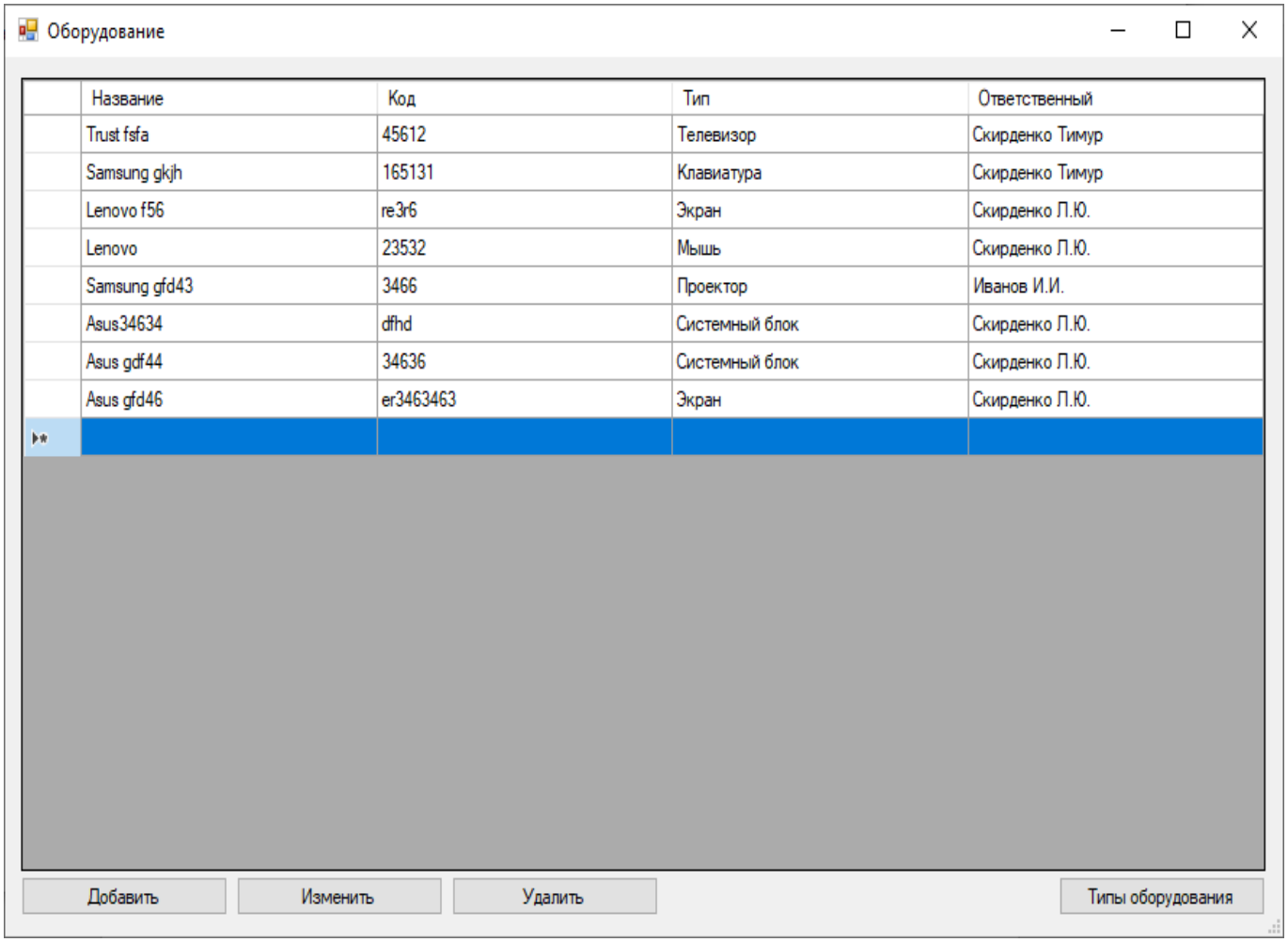

Рисунок 5.8 — Вікно “Обладнання”

При натисканні кнопки Додати відкривається вікно, зображене на рисунку 5.9.
В даному вікні є поля Назва, Код, Тип, Відповідальний. Поле Тип відображається у вигляді випадаючого списку, в якому можна вибрати потрібний тип обладнання.

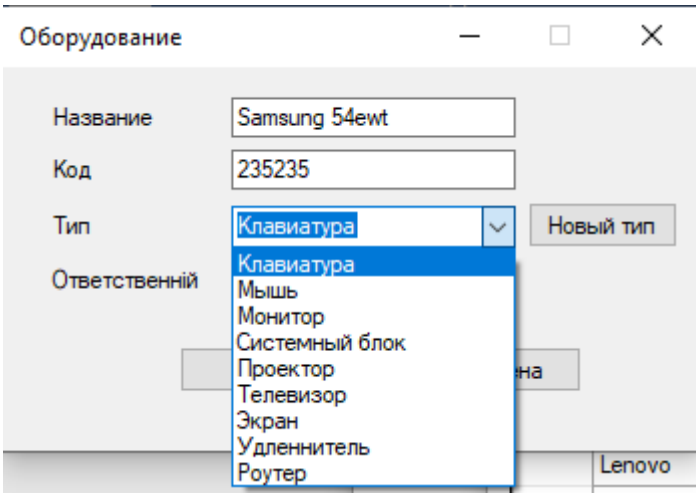

Рисунок 5.9 — Вікно додавання обладнання

Якщо потрібний тип відсутній в системі, його можна добавити, натиснувши кнопку “Новий тип”, в результаті чого відкриється вікно, зображене на рисунку 5.10.

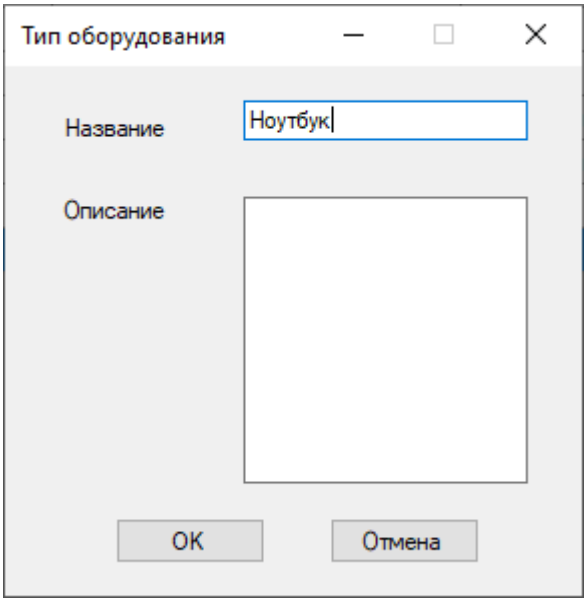

Рисунок 5.10 — Вікно додавання типу обладнання

Весь список типів обладнання, а також пошук обладнання конкретного типу можна здійснити в вікні (рисунок 5.11), яке відкривається натисканням кнопки “Типи обладнання”. Даний список можна доповнювати, редагувати та видаляти з нього непотрібні типи. Для цього доступні кнопки Додати, Змінити, Видалити.

В правій частині вікна відображається весь список обладнання вибраного типу.

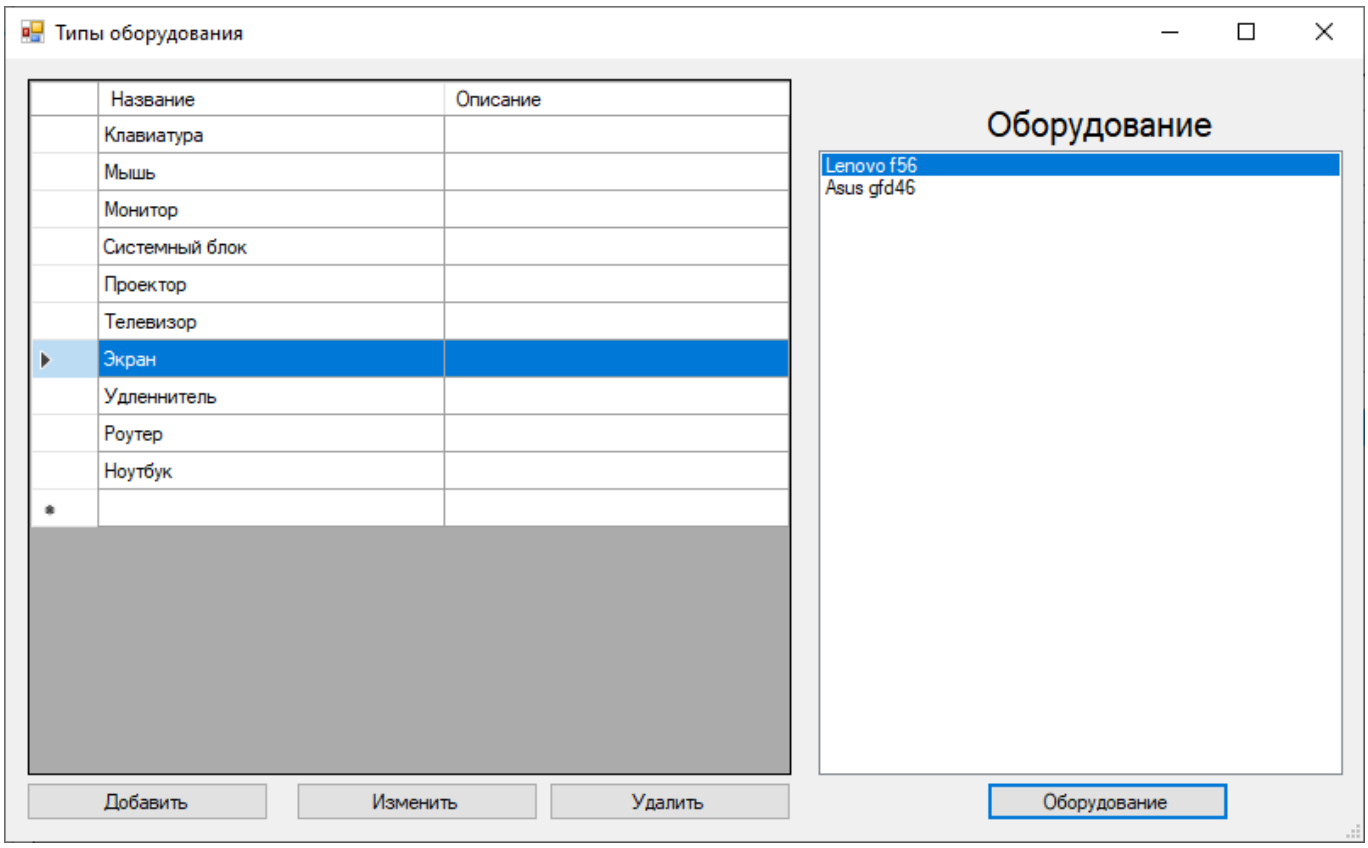

Рисунок 5.11 — Вікно “Типи обладнання”

Для видалення існуючого типу обладнання необхідно виділити потрібний запис і натиснути кнопку Видалити.

ВИСНОВКИ

1. В ході виконання даної роботи було розроблено систему автоматизації діяльності навчально-наукових лабораторій.

   Додаток було написано на мові програмування C# з використанням графічного інтерфейсу Windows Forms та СУБД Microsoft SQL Server. Програму можна використовувати для фіксації діяльності навчально-наукових лабораторій, автоматизації ведення документації.
2. В ході роботи було проведено огляд та зроблено аналіз засобів, що були використані для створення даного програмного забезпечення (середовища розробки Visual Studio 2019, інтерфейсу Windows Forms, СУБД Microsoft SQL Server та відповідних бібліотек мови C#).

   Для роботи з даним програмним забезпеченням необхідний лише комп’ютер середньої потужності.

   Користувач має змогу власноруч вводити дані про співробітників, кафедри, обладнання. Користувач може вести документацію більш швидко. Користувачеві надається можливість формувати задачі для співробітників лабораторії та відслідковувати їх виконання.

   Ця програма полегшує роботу навчально-наукових лабораторій та допомагає уникнути багатьох помилок та недоліків, які можуть виникати при кропітких задачах, які описані вище.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сайт Time Doctor // Таск менеджер: как выбрать? [Електронний ресурс] — Режим доступу: https://biz30.timedoctor.com/ru/task-manager/
2. Гриднева Н.Н. Проблеми науки і освіти на сучасному етапі суспільного розвитку // – Санкт-Петербург: Астеріон, 2011. – 128с
3. Роберт М. Чиста архітектура. Мистецтво розробки програмного забезпечення // – Санкт-Петербург: Пітер, 2016. – 352с
4. Майк Снайдер, Джим Стегер, Брендан Ландерс // Microsoft Dynamics CRM 2011 // – Москва: Еком, 2011. – 528с
5. Брайан Клифтон // Google Analytics для профессионалов // – Санкт-Петербург: Діалектика, 2013. – 608с
6. Сайт Professor Web // Работа с Visual Studio [Електронний ресурс] — Режим доступу: https://professorweb.ru/my/csharp/charp_theory/level2/2_4.php
7. Сайт METANIT.COM // Введение в Windows Forms [Електронний ресурс] — Режим доступу: https://metanit.com/sharp/windowsforms/1.1.php
8. Сайт Microsoft // System.Data.Entity Namespace [Електронний ресурс] — Режим доступу: https://docs.microsoft.com/en-us/dotnet/api/system.data.entity
9. Сайт Microsoft // Документация по SQL Server [Електронний ресурс] — Режим доступу:https://docs.microsoft.com/ru-ru/sql/sql-server/sql-server-technical-documentation?view=sql-server-2017&viewFallbackFrom=sql-server-2017
10. Энтони М. SQL. Сборник рецептов / Молинаро Энтони // SQL. Сборник рецептов / Молинаро Энтони. – Санкт-Петербург: Символ-плюс, 2009. – С. 116.

ДОДАТОК 1

Система автоматизації діяльності
навчально-наукових лабораторій

Специфікація

УКР. КПІ ім. Ігоря Сікорського_ТР62148_20Б

Аркушів 2

Київ 2020

Позначення	Найменування	Примітки
Документація		
УКР. КПІ ім. Ігоря Сікорського_ТР62148_20Б	Записка.docx	Пояснювальна записка
Компоненти		
УКР. КПІ ім. Ігоря Сікорського_ТР62148_20Б	**DocumentEdit.cs**	модуль пошуку та зберігання документів
УКР. КПІ ім. Ігоря Сікорського_ТР62148_20Б	**DocumentSearch.cs**	модуль пошуку та зберігання документів
УКР. КПІ ім. Ігоря Сікорського_ТР62148_20Б	**FileAttach.cs**	модуль пошуку та зберігання документів
УКР. КПІ ім. Ігоря Сікорського_ТР62148_20Б	**TaskCreator.cs**	модуль моніторингу виконання задач
УКР. КПІ ім. Ігоря Сікорського_ТР62148_20Б	**TaskCreatorFromFile.cs**	модуль моніторингу виконання задач

УКР. КПІ ім. Ігоря Сікорського_ТР62148_20Б	**CheckTask.cs**	модуль моніторингу виконання задач
УКР. КПІ ім. Ігоря Сікорського_ТР62148_20Б	**FindTask.cs**	модуль моніторингу виконання задач

ДОДАТОК 2

Система автоматизації діяльності
навчально-наукових лабораторій

Лістинг програми

УКР. КПІ ім. Ігоря Сікорського_ТР62148_20Б

Аркушів 3

```
//Метод зчитування та запису файлу в базу даних
private void button4_Click(object sender, EventArgs e)
        {
            int id = int.Parse(IdFile.Text);
            Document document = db.Documents.Find(id);
            Stream fs = new FileStream(@"D:\RTFilefile.doc", FileMode.CreateNew);
            fs.Write(document.File, 0, document.File.Length);
            fs.Close();
            System.Diagnostics.Process.Start(@"D:\RTFilefile.doc").WaitForExit();
            fs = new FileStream(@"D:\RTFilefile.doc", FileMode.Open);
            document.File = new byte[fs.Length];
            fs.Read(document.File, 0, (int)fs.Length);
            fs.Close();
            db.Entry(document).State = EntityState.Modified;
            db.SaveChanges();
            File.Delete(@"D:\RTFilefile.doc");
        }
//Метод визначення користувача
private void button1_Click(object sender, EventArgs e)
        {
            var user = db.Users.Where(u => u.Login == textBox1.Text && u.Password == 
textBox2.Text);
            if (user.Count() > 0)
            {
                Owner = user.FirstOrDefault();
                Hide();
                Form1 mainForm = new Form1(Owner);
                mainForm.ShowDialog();
```

```
            Close();
        }
        else
        {
            MessageBox.Show("Пользователь не найден");
        }
    }

//Метод створення нової задачі
private void button1_Click(object sender, EventArgs e)
    {
        TaskForm taskForm = new TaskForm();
        List<Employee> employees = db.Employees.ToList();
        taskForm.Employee.DataSource = employees;
        taskForm.Employee.DisplayMember = "Name";
        List<TaskStatus> taskStatuses = db.TaskStatuses.ToList();
        taskForm.Status.DataSource = taskStatuses;
        taskForm.Status.ValueMember = "Id";
        taskForm.Status.DisplayMember = "Name";
        DialogResult result = taskForm.ShowDialog(this);
        if (result == DialogResult.Cancel) return;
        UsrTask usrTask = new UsrTask();
        usrTask.Name = taskForm.Name.Text;
        usrTask.StartDate = taskForm.StartDate.Value.Date;
        usrTask.EndDate = taskForm.EndDate.Value.Date;
        usrTask.Employee = (Employee)taskForm.Employee.SelectedItem;
        usrTask.TaskStatus = (TaskStatus)taskForm.Status.SelectedItem;
        db.UsrTasks.Add(usrTask);
        db.SaveChanges();
```

```
            MessageBox.Show("Новая задача добавлена");
        }
//Метод оновлення документу
private void Button2_Click(object sender, EventArgs e)
        {
            if (dataGridView1.SelectedRows.Count > 0)
            {
                int index = dataGridView1.SelectedRows[0].Index;
                int id = 0;
                bool converted = int.TryParse(dataGridView1[0, index].Value.ToString(), out
id);
                if (converted == false) return;
                Document document = db.Documents.Find(id);
                DocumentForm documentForm = new DocumentForm();
                documentForm.button4.Visible = true;
                documentForm.IdFile.Text = document.Id.ToString();
                documentForm.Name.Text = document.Name;
                List<DocumentType> departments = db.DocumentTypes.ToList();
                documentForm.Type.DataSource = departments;
                documentForm.Type.ValueMember = "Id";
                DialogResult result = documentForm.ShowDialog(this);
                document.ModifiedOn = DateTime.Now;
                document.ModifiedBy = int.Parse(user.Text);
                document.DocumentType = (DocumentType)documentForm.Type.SelectedItem;
                db.Entry(document).State = EntityState.Modified;
                db.SaveChanges();
                MessageBox.Show("Документ обновлен");
            }
        }
```

ДОДАТОК 3

Система автоматизації діяльності
навчально-наукових лабораторій

Опис програмного коду

УКР. КПІ ім. Ігоря Сікорського_ТР62148_20Б

Аркушів 2

Київ 2020

АНОТАЦІЯ

Додаток містить опис .cs файлів модуля пошуку та зберігання документів та модуля моніторингу виконання задач. Ці модулі написані мовою програмування C# з використанням технології WinForms, яка дозволяє працювати з додатком та виводити всі результати роботи додатку .

В додатку наведено опис .cs файлів двох модулів, що були розроблені в ході дипломної роботи

1.1 DocumentEdit.cs

Файл містить логіку відкриття документу з бази даних, можливість його редагування та збереження назад до бази даних.

1.2 DocumentSearch.cs

Файл містить логіку пошуку документу, групування документів, сортування документів за умовою у базі даних. Результат пошуку виводиться до форми DocumentMainForm.cs.

1.3 FileAttach.cs

Файл містить логіку, яка надає можливість вибрати потрібний файл на комп’ютері та завантажити його до бази даних. Прикріплений файл та його характеристики відображаються у формі DocumentForm.cs.

1.4 TaskCreator.cs

Файл містить логіку створення задач через форму TaskCreatorForm.cs. Введені користувачем дані валідуються та зберігаються до бази даних.

1.5 TaskCreatorFromFile.cs

Файл містить логіку створення задач шляхом завантаження потрібного файлу до бази даних. Файл проходить парсинг, далі в базі даних з’являється задача.

1.6 CheckTask.cs

Файл містить логіку, яка надає можливість перевірити стадію виконання задачі певного користувача.

1.7 FindTask.cs

Файл містить логіку пошуку задач, групування задач та сортування задач за умовою в базі даних. Результат пошуку виводиться до форми TaskMainForm.cs.